ARM Cortex™-A15 MPCore - NEON

Product Revision r3

Software Developers Errata Notice

Non-Confidential - Released

This document contains all software developer errata known at the date of issue in releases r3p0 up to and including revision r3p3

ARM®

Software Developers Errata Notice

Non-Confidential Proprietary Notice

Web Address

http://www.arm.com

Feedback on content

If you have any comments on content, then send an e-mail to errata@arm.com . Give:

- the document title
- the document number, ARM-EPM-028093
- the page numbers to which your comments apply
- a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Release Information

Errata are listed in this section if they are new to the document, or marked as "updated" if there has been any change to the erratum text in Chapter 2. Fixed errata are not shown as updated unless the erratum text has changed. The summary table in section 2.2 identifies errata that have been fixed in each product revision.

21 Mar 2013: Changes in Document v5

Page	Status	ID	Cat	Rare	Summary of Erratum
12	New	801819	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing
17	Updated	798181	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

14 Feb 2013: Changes in Document v4

There are no New or Updated errata associated with this release.

31 Jan 2013: Changes in Document v3

Page	Status	ID	Cat	Rare	Summary of Erratum
11	New	799271	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock
17	New	798181	CatB	Rare	Moving a virtual page that is being accessed by an active process can lead to unexpected behavior
19	Updated	763126	CatB	Rare	Three processor exclusive access livelock

15 Nov 2012: Changes in Document v2

No new or updated errata. REVID[0] used to indicate an ECO Patch for Defect 794724.

23 Oct 2012: Changes in Document v1

Page	Status	ID	Cat	Rare	Summary of Erratum
10	New	794724	CatA		L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time
19	Updated	763126	CatB	Rare	Three processor exclusive access livelock

Contents

Chapter 1.
Introduction

This chapter introduces the errata notice for the ARM Cortex-A15 processor.

1.1. Scope of this document

This document describes errata categorized by level of severity. Each description includes:

- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behavior occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible

This document describes errata that may impact anyone who is developing software that will run on implementations of this ARM product.

1.2. Categorization of errata

Errata recorded in this document are split into the following levels of severity:

Table 1 Categorization of errata

Errata Type	Definition
Category A	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
Category A(rare)	A critical error. No workaround is available or workarounds are impactful. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category B	A significant error or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
Category B(rare)	A significant error or a critical error with an acceptable workaround. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category C	A minor error.

Chapter 2.
Errata Descriptions

2.1. Product Revision Status

The r*n*p*n* identifier indicates the revision status of the product described in this book, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

2.2. Revisions Affected

Table 2 below lists the product revisions affected by each erratum. A cell marked with **X** indicates that the erratum affects the revision shown at the top of that column.

This document includes errata that affect revision r3 only.

Refer to the reference material supplied with your product to identify the revision of the IP.

Table 2 Revisions Affected

ID	Cat	Rare	Summary of Erratum	r3p0	r3p1	r3p2	r3p3
784421	CatA		CPU may deadlock if short interrupt pulse deasserts just as a WFI instruction is executed	X			
788420	CatA		A15 can stall if 16 stores are issued and the 16th store gets its BRESP before any of the earlier stores get their BRESP	X	X		
794724	CatA		L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time	X	X	X	
799271	CatA		A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock	X	X	X	
801819	CatA	Rare	An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing	X	X	X	
784420	CatB		Speculative instruction fetches with MMU disabled might not comply with architectural requirements	X	X	X	X
784477	CatB		CTIINTACK register needs clearing each time it is set	X	X	X	
785769	CatB		Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode	X	X	X	X
798181	CatB		Moving a virtual page that is being accessed by an active process can lead to unexpected behavior	X	X	X	
763126	CatB	Rare	Three processor exclusive access livelock	X	X	X	X
773023	CatC		Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI	X	X	X	X
777769	CatC		ICache parity error may not be corrected for NC code	X	X	X	X

ID	Cat	Rare	Summary of Erratum	r3p0	r3p1	r3p2	r3p3
784419	CatC		An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt	X	X	X	
784469	CatC		CTI Authentication Status register is incorrect	X	X	X	
788419	CatC		If a single memory address is aliased as both shareable and non-shareable, a CMO to this same line might deadlock	X	X	X	

r3p2 implementation fix

Note the following errata may be fixed in some implementations of r3p2. This can be determined by reading the REVIDR register where a set bit indicates that the erratum is fixed in this part.

REVIDR[0]	794724 L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time
REVIDR[1]	799271 A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock
REVIDR[2]	Not used in product revision r3pX
REVIDR[3]	801819 An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing
REVIDR[4]	798181 Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

Note that there is no change to the MIDR which remains at r3p2, but the REVIDR might be updated from 0x00 to indicate that one or more errata are corrected as shown above. Software will identify this release through the combination of MIDR and REVIDR.

2.3. Category A

784421: CPU may deadlock if short interrupt pulse deasserts just as a WFI instruction is executed

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0. Fixed in r3p1.

Description

If an interrupt is asserted to a CPU and then deasserted before that CPU takes the interrupt and that CPU is executing a WFI instruction at the same time, it is possible that the CPU will enter WFI state and never wake up.

Note: This erratum matches bug #5362 in the ARM internal Jira database.

Conditions

The erratum can occur only if the following sequence of conditions is met:

1) The CPU starts to execute a WFI instruction, with no interrupt pending.
2) An interrupt is asserted. This can be a physical or a virtual interrupt, from an interrupt pin or from the internal GIC.
3) The Interrupt is deasserted without being taken by the CPU.
4) The CPU completes execution of the WFI instruction and enters WFI mode.

Implications

If the above conditions occur, it is possible that the CPU will never wake up from WFI mode. Subsequent interrupts to the CPU will be ignored.

Workaround

There is no useful workaround.

788420: A15 can stall if 16 stores are issued and the 16th store gets its BRESP before any of the earlier stores get their BRESP

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1. Fixed in r3p2.

Description

A15 issues 15 writes on the AXI write channel without receiving a BRESP for any of them. A15 then issues a 16th write and receives a BRESP for the 16th write before receiving a BRESP for any of the first 15 writes. If this occurs, A15 might deadlock.

Note: This erratum matches bug #5380 in the ARM internal Jira database.

Conditions

1) A15 issues 16 stores on the AXI write channel without receiving a BRESP.

2) A BRESP is returned for the 16th store before a BRESP is received for any of the earlier stores.

Implications

If the above conditions occur A15 might deadlock.

Workaround

There is no software workaround.

794724: L2 Cache initialization sequence may not function correctly if the L2 tag RAM requires two-cycle setup time

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

If a Cortex-A15 MPCore implementation uses an L2 tag RAM that requires a two cycle setup time for address and data, the L2 cache might not be correctly invalidated by the hardware initialization sequence that occurs after the deassertion of nL2RESET.

By default, the Cortex-A15 MPCore L2 tag RAM input paths are single cycle paths for both setup and hold. However, if the L2 tag RAM used in an implementation requires it, software can program L2CTLR[9] to 1 to configure the setup paths to be two cycle multicycle paths. These must be set correctly before the data cache is enabled.

Cortex-A15 MPCore initializes the L2 cache in hardware when the L2 is reset. Because this hardware initialization occurs before the L2CTLR register can be programmed by software, the hardware sequence must assume tag RAM array timings that will work with all legal RAM instances.

During hardware RAM initialization of the L2 on affected versions of Cortex-A15 MPCore the setup used for the L2 tag RAM is a single cycle. It should be two cycles to support RAMs that require more setup.

Note: This erratum matches bug #5403 in the ARM internal Jira database.

Configurations Affected

This erratum affects implementations that require L2CTLR[9] to be set to 1 because the L2 tag RAM requires two cycle setup.

This erratum does not affect the processor if the revision and variant reported by the MIDR is r3p0, r3p1 or r3p2 and the REVIDR[0] is set to 1.

Conditions

This erratum requires the following condition:

- The implementation uses an L2 tag RAM that requires two cycle setup on address and data. Affected implementations will program L2CTLR[9]=1 before enabling the data cache.

Implications

The L2 Cache may not be correctly initialized after deassertion of nL2RESET.

1) Valid lines with unknown addresses and data might be present in the cache after reset. If the valid lines match any address in physical memory that is accessed before the lines are evicted, they could deliver incorrect data on reads.

2) If the lines appear valid and dirty, they could generate spurious memory transactions that would corrupt memory or generate errors in the system.

Workaround

Using a lower frequency such that the tag RAM instance can meet timing without the two cycle setup is a valid system workaround. The lower frequency need only be used between deassertion of nL2RESET and the completion of the cache initialization sequence. Software can ensure that the hardware initialization sequence has completed by executing any data cache maintenance operation (e.g. DCCISW) followed by a DSB. The maintenance operation will not complete until the L2 initialization is complete.

Software initialization of the L2 cache is not a valid workaround. The DCISW invalidate by set/way instruction is treated by A15 as a DCCISW clean/invalidate by set/way instruction. At the end of a software initialization of the L2 cache, the cache would be correctly invalid, but the software initialization could cause spurious evictions of lines incorrectly marked as valid and dirty.

799271: A combination of Non-cacheable or streaming writes, a DSB instruction, and a snoop can cause a core to deadlock

Category A

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

If a certain combination of snoops, DSB instruction execution, and stores occurs with a particular timing, a Cortex-A15 MPCore CPU can deadlock. The deadlocked CPU will be unable to complete an eviction until earlier writes complete, and those writes will not be able to complete until that eviction completes.

Note: This erratum matches bug #5416 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r3p0, r3p1 or r3p2 and REVIDR[1] is set to 1.

Conditions

1) One core (coreX) is executing store instructions that are not allocating to the L1 cache. These will be a combination of:
   - Writes to Non-cacheable, Strongly-ordered, or Device memory locations.
   - Write-streaming full line cacheable writes.
2) These stores back up in the memory system and allocate 12 entries of the L2 Write Request Queue
3) CoreX begins the eviction of a cache line (A) that is being replaced by a returning cache line fill
4) Another core (coreY) executes a DSB instruction that generates a TLB synchronization request to coreX
5) A memory request occurs (from coreZ, an ACP request, an external snoop, or a translation table walk request) that
   - hits the cache line (A) being evicted from coreX, OR
   - hits another line in the L1 data cache of coreX that is in uniqueClean or uniqueDirty state.

Implications

If the above scenario occurs with a particular timing, it is possible that the eviction from coreX cannot complete until another L2 write buffer credit is returned, and no such credit can be returned until the eviction completes. In this situation, the processor deadlocks.

In a single core configuration without ACE, the erratum cannot not occur.

In a single core configuration with ACE, but with no other ARM cores capable of issuing a “DVM Sync” request, the erratum cannot occur.

In a two core configuration without ACE, the erratum is believed to be very rare.

In a two core configuration with ACE, or a three or more core configuration, the erratum can occur.

Workaround

There is no known workaround.

2.4. Category A (Rare)

801819: An eviction from L1 data cache might stall indefinitely in the L2 write buffer preventing a snoop from completing

Category A Rare

Products Affected: Cortex-A15 MP Core -NEON.
Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

A livelock can occur in the L2 cache arbitration that might prevent a snoop from completing. Under certain conditions this can cause the system to deadlock.

When a cacheable store is executed in the L2 it requires an entry to be allocated in the Fill Evict Queue (FEQ). If an entry is not available the store will restart, as will any younger stores from the same CPU, because stores from the same CPU must execute in order.

If two cacheable stores from the same CPU issue into the L2 just as an FEQ entry becomes available, it is possible for the first store (ST1) to detect the FEQ as full, but the second store (ST2) to see an available FEQ entry and allocate that entry. Both stores are then restarted, the ST1 due to the lack of an FEQ entry, ST2 because the ST1 restarted. This means the FEQ entry temporarily allocated by ST2 is marked for deallocation.

The two stores will then reissue to the L2. If the timing of the reissue aligns with the deallocation of the FEQ entry temporarily allocated previously by ST2, it is possible that ST1 will again detect an FEQ full condition and ST2 will see an FEQ entry available and allocate it. If there is no other activity in the L2 to change the timings, this cycle can repeat until a second FEQ entry becomes available and both stores can proceed, or another FEQ entry is allocated for an unrelated request and both stores stop reissuing waiting for an FEQ entry to become available.

Depending on the L2 cache configuration options there might need to be one or more unrelated stores between ST1 and ST2 in order to hit the required timing for the livelock.

If there is an eviction in the write buffer that is behind the stores then that eviction might be stalled as long as the livelock continues. This stalled eviction might stall a snoop hit on the L1 data cache of that CPU. The stalled snoop might block further FEQ entries from deallocating, either through a dependency in the ACE memory system, or because the FEQ is full of snoop hits to the L1 data cache of that CPU. If all of these conditions occur the system can deadlock.

Note: This erratum matches bug #5433 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r3p0, r3p1, or r3p2 and REVIDR[3] is set to 1.

Conditions

1) The L2 Fill Evict Queue (FEQ) is full.
2) Cacheable write ST1 issues followed by cacheable write ST2 from the same CPU.
3) An eviction is waiting in the write buffer behind ST1, ST2, and one or more additional stores.
4) A snoop (from another A15 core or from the ACE AC channel) is waiting for that eviction to complete.
5) That snoop is preventing further FEQ entries from deallocating.
6) Very specific timing conditions.

Implications

If the erratum conditions are met the part will deadlock.

Cortex-A15 will only issue two types of cacheable stores from a CPU into the L2: Write-Back No-Allocate stores, or Write-Back stores that have been gathered into a full line write using the L1 write streaming logic.

For implementations of Cortex-A15 configured with the “L2 arbitration register slice” (arb-slice) option (typically four core systems) the only cacheable stores that can cause the issue are Write-Back No-Allocate stores. Because Write-Back No-Allocate stores are not commonly used and are easy to avoid, there is a straightforward workaround for these implementations. For configurations with the arb-slice, this erratum is Category B.

For implementations of Cortex-A15 configured without the arb-slice option (typically 1 or 2 core systems) Write-Back No-Allocate stores and streaming cache line writes can both lead to the deadlock, although hitting the conditions with

streaming cache line writes is much more difficult. There is still a full workaround, but because it involves disabling write-streaming there is a performance hit on some streaming memory workloads. For configurations without the arb-slice, this erratum is Category A Rare.

Workaround

Do both of the following:

1) Do not use the write-back no-allocate memory type.

2) Do not issue write-back cacheable stores at any time when the cache is disabled (SCTLR.C=0) and the MMU is enabled (SCTLR.M=1). Because it is implementation defined whether cacheable stores update the cache when the cache is disabled it is not expected that any portable code will execute cacheable stores when the cache is disabled.

For implementations of Cortex-A15 configured without the “L2 arbitration register slice” option (typically one or two core systems), you must also do the following:

3) Disable write-streaming in each CPU by setting ACTLR[28:25] = 0b1111

2.5. Category B

784420: Speculative instruction fetches with MMU disabled might not comply with architectural requirements

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2, r3p3

Description

When all applicable stages of translation are disabled, an ARMv7 processor must follow some architectural rules regarding speculative fetches and the addresses to which these can be initiated. These rules avoid potential reads to read-sensitive areas. For more information about these rules see the description of "Behavior of instruction fetches when all associated MMUs are disabled" in the ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition. Cortex-A15 normally operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the BTB (branch target buffer) will contain branch predictions. If both stages of translation are then disabled, but branch prediction remains enabled, these stale BTB entries can cause A15 to violate the rules for speculative fetches.

Note: This erratum matches bug #5360 in the ARM internal Jira database.

Conditions

The erratum can occur only if the following sequence of conditions is met:

1) MMU enabled for at least one stage of address translation
2) Branch prediction enabled
3) Branches executed
4) MMU disabled for all applicable stages of address translation

Note: When executing in a Non-secure PL1 or PL0 mode, for condition 1 at least one stage of address translation must be enabled, and for condition 4 both stages of address translation must be disabled. When executing in any other mode, there is only one stage of address translation, that must be enabled for condition 1 and disabled for condition 4.

Implications

If the above conditions occur, it is possible that after the MMU is disabled, speculative instruction fetches will occur to read-sensitive locations.

Workaround

Branch prediction should be disabled when the MMU is disabled after having been enabled. This should be done by clearing the appropriate Z bit in the System Control register at the same time as or just before the final stage of translation is disabled. Branch prediction should remain disabled until the MMU is enabled, or until the BTB has been flushed. On A15, the BPI* branch predictor maintenance commands will not invalidate the BTB. The BTB can be flushed by setting bit 0 of the ACTLR register, doing any instruction cache invalidate instruction (e.g. ICIALLU), and then clearing bit 0 of the ACTLR register.

784477: CTIINTACK register needs clearing each time it is set

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

The CTI contains a CTIINTACK register, which enables a trigger to be acknowledged through software, instead of using a hardware knowledge using the CTITRIGOUTACK input. The correct operation of this register is that writing a one to the bit corresponding to a trigger output will cause that trigger to be cleared, and this will not affect future triggers.

Because of this erratum, when a bit in the CTIINTACK register is set, it remains set until cleared by writing zero to the register. This causes the corresponding trigger outputs to be acknowledged immediately if they occur again, which can lead to them being missed.

The CTIINTACK register is normally used in two cases:

- To clear a debug-originated interrupt, if required by the interrupt controller.
- To clear a debug entry request generated by another processor, when cross-halting is used.

Conditions

The following conditions must occur:

- A CTI trigger output fires.
- The CTI CTIINTACK register is used to acknowledge the trigger output, by writing a one to the bit corresponding to that trigger output.
- The same trigger output fires again before the corresponding bit in the CTIINTACK register is cleared.

Implications

Trigger outputs might be missed:

- In the case of a debug-originated interrupt that uses CTIINTACK to clear the interrupt, events other than the first event might not cause an interrupt to occur.
- In the case of a cross-halting debug request, after the first time a processor halts and restarts, it might halt without halting other processors with it.

Workaround

This is a workaround for tools vendors.

When the CTIINTACK register is written with a nonzero value, it must be immediately written to again with the value zero. This prevents any future events on the corresponding trigger output from being acknowledged.

If this workaround is used, there remains a race condition, whereby a trigger output occurring between the two register writes might be lost. This is in general not significant, because the timing of trigger outputs and the timing of register writes are not highly correlated, and if the trigger output had occurred before the first register write, then it would also have been lost.

785769: Undefined exception is not generated for LDC/STC instructions which access DCC registers in User mode

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2, r3p3

Description

The Debug Communication Channel (DCC) registers are accessible using MCR/MRC instructions. LDC/STC instructions provide alternate access to DCC registers DBGDTRTXint/DBGDTRRXint.

In Non-debug state when DBGDSCR.UDCCdis is set to 1, then access to DCC register using MCR/MRC and LDC/STC instructions from User mode should generate an Undefined Instruction exception. Access using MCR/MRC instructions correctly generates an Undefined Instruction exception correctly. However access using LDC/STC instructions does not generate the Undefined Instruction exception and incorrectly accesses the register.

Note: This erratum matches bug #5372 in the ARM internal Jira database.

Conditions

1) The processor is in Non-debug state and User mode.
2) DBGDSCR.UDCCdis is set to 1.
3) Either the Hypervisor trap to debug registers in not set (HDCR.TDA==0) or the processor is in Secure state.
4) LDC to DBGDTRTXint or STC to DBGDTRRXint is executed.

Implications

If LDC or STC instructions are executed in User mode, then DCC traffic between debug host and debug target from FIQ/IRQ/Supervisor/Monitor/Abort/Hypervisor/Undefined/System modes can be corrupted due to this errata.

Workaround

Tools must use MCR/MRC instructions to access Debug Communication Channel (DCC) registers from User mode, instead of LDC/STC instructions, in Non-debug state.

Alternatively, software can avoid corruption of DCC traffic by Non-secure User mode code by setting the hypervisor trap for debug register accesses (HDCR.TDA), and handling the LDC/STC instruction appropriately in hypervisor code. There is no software workaround to prevent LDC/STC instructions executing in Secure User mode from corrupting DCC traffic handled in other processor modes.

798181: Moving a virtual page that is being accessed by an active process can lead to unexpected behavior

Category B

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

A TLB invalidate instruction followed by a DSB instruction to ensure its completion should remove all uses of the old translation from the system. On Cortex-A15 this might not occur in the following cases:

- The exclusive monitor is physical address based. Moving a memory region in physical memory might cause unexpected results, including multiple threads acquiring the same lock.
- Hazarding logic to guarantee read after read ordering to the same address is physical address based. If a memory region is moved in physical memory ordering violations might occur.
- The DSB instruction might complete before all memory transactions to the invalidated translation are globally observed.

The first two conditions might lead to unexpected ordering and Load-Exclusive/Store-Exclusive instructions behaving in an unexpected way. The third condition might lead to data corruption if the software attempts to access or allows access to the physical memory of the invalidated or moved region before the memory transactions have completed.

Note: This erratum matches bugs #5405, #5407, #5428, #5429 in the ARM internal Jira database.

Configurations Affected

This erratum does not affect the processor if the revision and variant reported by the MIDR is r3p0, r3p1 or r3p2 and REVIDR[4] is set to 1.

Conditions

1) Software modifies the translation tables to invalidate, restrict the permissions of, or change the physical address of a page or block.
2) Software executes a TLB invalidate instruction to invalidate any TLB entries holding the previous translation.
3) Software executes a DSB instruction to ensure completion of the TLB invalidate instruction and global observation of any memory transactions that depended on the previous translation.

Implications

When software changes the translation tables and invalidates a modified TLB entry:

1) exclusive monitors in the system might be using the old translation
   - this might lead to unexpected Load-Exclusive/Store-Exclusive behavior.
2) the read after read hazard logic might be using the old translation
   - this might lead to a load instruction returning older data than a previous load instruction that accessed the same virtual memory location.
3) outstanding loads or stores to the old translation might not be globally observed
   - this might lead to data corruption if the physical memory of the invalidated memory region is accessed before the memory requests that used the old translation are complete.

Workaround

To ensure the completion of a TLB invalidate, software must first follow the translation table invalidation steps specified in the ARM Architecture Reference Manual:

1) Modify the translation tables in memory as required.

2) Execute a DSB to ensure observation of the stores to the translation tables.

3) Execute one or more TLB invalidate instructions targeting the affected memory regions.

4) Execute a DSB to ensure the TLB invalidate instructions from step 3 have been propagated to all processors in the system.

Next, on the processor that is performing the translation table maintenance, it must also perform these two additional steps:

5) Execute a dummy TLBIMVAIS, meaning a TLBIMVAIS to any address with any ASID.

6) Execute a DSB instruction to ensure the dummy TLB invalidate instruction has been propagated to all processors in the system.

Finally, identify any Cortex-A15 processors in the system that are currently using the translation context of the region being invalidated. For global regions this will be all Cortex-A15 processors not powered down or in reset. For non-global regions this will be all Cortex-A15 processors that are currently using the same translation context (typically ASID and VMID) as the modified translation table entries. On each of those processors, ensure that the following occur (in any order):

- execution of a CLREX instruction
- execution of a DMB or DSB instruction

Once the above sequence is complete all uses of the old translation in the system will removed, and any memory transactions to the old translation will be globally observed.

To meet the final requirement, the processor performing the translation table maintenance must execute a CLREX and a DMB or DSB instruction.

The easiest way to meet the final requirement on other processors in the system is to send an inter-processor interrupt to each required processor. Most interrupt handlers will execute a CLREX instruction (as required by the ARM Architecture on a context switch). The interrupt handler must execute a DMB instruction and then signal completion to the processor doing the translation table maintenance.

Note: this workaround must be implemented on any ARM Architecture processor that is executing TLB invalidate instructions that may affect a Cortex-A15 processor. This would include translation table maintenance being performed on a Cortex-A7 processor when a Cortex-A15 processor is present in the system..

2.6. Category B (Rare)

763126: Three processor exclusive access livelock

Category B Rare
Products Affected: Cortex-A15 MP Core -NEON.
Present in: r3p0, r3p1, r3p2, r3p3.

Description

In a system with three or more coherent masters that all use the ldrex/strex synchronization primitives to access a semaphore in coherent cacheable memory, there is a possibility of a livelock condition where two masters continuously attempt and fail to get the lock while the third master continuously reads the lock.

This erratum is heavily dependent on a unique set of initial conditions, and upon specific interconnect timing once the livelock has started. It is expected to be rare in a real system that the timing conditions will be hit.

An example: two cores C1 and C2 are contending for a lock using ldrex/strex, and core C3 is looping reading the same semaphore location. Once the livelock condition has started, from the perspective of C1, the sequence will look like this:

1) Execute ldrex, hits the cache in unique state.
2) External snoop takes line to shared state (triggered by C3 read).
3) Execute instructions to process the ldrex result and prepare the strex data.
4) Execute strex, hits cache shared, issues readUnique to bring in line unique.
5) External snoop invalidates line, clearing monitor (triggered by C2 strex that will eventually fail the monitor).
6) Line returns in unique state, but strex fails due to cleared monitor.
7) Loop back to step 1.

C1 and C2 constantly issue ReadUniques due to failing store exclusives that invalidate the line in the other core, each core causing the others strex to fail without making forward progress. No forward progress is made until/unless one of the cores stops (possibly due to an interrupt) or interconnect timing happens to allow enough time for one of them to complete.

NOTE: this erratum is describing additional limitations on exclusives loads and stores in a multi-core system including A15. There is no plan to fix this erratum on future A15 cores, as reasonable code following the ARM architecture guidelines should not be affected.

NOTE: This erratum matches bug #4637 in the ARM internal Jira database.

Conditions

1) One master continuously reading the location of the semaphore.
2) Two masters doing a ldrex/strex loop to the semaphore.
3) Semaphore in write-back shared memory.
4) Three master system (3+ core A15, or 3+ total processors in the system over ACE).

Implications

Neither C1 nor C2 will ever succeed in gaining the lock. Software could stop making progress. An interrupt to one of the cores C1/C2/C3 would likely break the livelock.

Workaround

If there are no more than two coherent masters in the system, no workaround is needed, the issue will not be seen.

The latest version of the ACE specification adds additional command types and system logic to allow processors to avoid this issue. This specification update was not available in time for A15 to take advantage of it and A15 does not implement this ACE feature. As an alternative, A15 installed hardware in each processor to detect that the load/store exclusive livelock scenario may be occurring and delay snoops for a period of time to allow the load exclusive/store exclusive loop to complete and make forward progress. With this fix, no existing code that uses ldrex/strex should need to be rewritten if it follows the ARM Architecture Reference Manual guidelines in the “A3.4 Synchronization and Semaphores” section and is not unreasonably long.

To enable this hardware on Cortex-A15 you must set the "Snoop-delayed exclusive handling" bit in the Auxiliary Control Register, ACTLR[31] to 1. The reset value of ACTLR[31] is 0 for all product revisions r2pX, r3p0, r3p1 and r3p2.

Note: all references to “ldrex” encompass all Load-Exclusive instructions and “strex” encompass all Store-Exclusive instructions.

2.7. Category C

773023: Order may not be maintained between Strongly Ordered memory requests and Device memory requests on ACE/AXI

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2, r3p3.

Description

A15 maintains order between Strongly Ordered (SO) memory requests as required by the ARM architecture memory ordering model. This is done inside the A15 by use of internal ordering logic. On the interconnect, A15 enforces ordering by using the same ARID/AWID value for all SO memory requests from a given processor (guaranteeing read/read and write/write ordering) and by waiting for the completion of all SO and Device memory requests on one channel before issuing SO or Device requests from the same core on the other channel (guaranteeing read/write and write/read ordering).

A15 does the same for Device memory.

However, A15 uses different ARID and AWID for Device memory requests from a given CPU and Strongly Ordered memory requests from the same CPU. Due to this fact, it is possible that the an SO read from a given CPU could pass a Device read from the same CPU and arrive at a single peripheral out of order.

Conditions

1) A system has memory mapped peripherals larger than 4KB
2) Some of the pages mapped to that peripheral are mapped Strongly Ordered and some are mapped Device
3) Software depends upon ordering of these Strongly Ordered and Device memory requests

Implications

This is not an issue for any device that fits in one 4KB memory page, as it is only possible to have a single memory type for that page (SO/Dev aliasing is not allowed).

For larger peripherals, it is possible that Strongly Ordered or Device transactions could arrive at the peripheral out of order.

Workaround

A given peripheral device should be mapped to all Strongly Ordered or all Device memory.

777769: ICache parity error may not be corrected for NC code

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2, r3p3.

Description

If an instruction fetch to a non-cacheable page in memory gets a false hit on a line in the instruction cache due to a parity error in the instruction cache tag array, incorrect instructions may be executed.

Note: This erratum matches bug #5312 in the ARM internal Jira database.

Conditions

1) MMU enabled
2) Instruction fetch to page marked SO/Dev/Normal-Non-Cacheable
3) Parity error in instruction cache tag
4) Corrupted tag matches the physical address of the cache line being fetched

Implications

In an A15 configured with L1 parity/ECC and with parity/ECC checking enabled, in very rare circumstances a parity error can cause delivery of bad instructions while executing non-cacheable code. This is not expected to be an issue in normal systems as no normal programs will have instructions in non-cacheable memory with the MMU enabled. At boot, or any other time that the MMU is disabled, the erratum will not occur.

Workaround

Place instructions in cacheable memory whenever possible. If you must run non-cacheable code with the MMU enabled, first invalidate the instruction cache.

784419: An unaligned load crossing a 4K boundary between SO/Dev memory and WB memory can stall the core until the next interrupt

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

If an unaligned load crosses a 4k page boundary between two pages where the lower page is mapped to Strongly-ordered or Device memory and the upper page is mapped to Normal, Write-Back Cacheable memory, in certain rare cases the core may stall until the next interrupt. A memory transaction that crosses a boundary between these page types is architecturally UNPREDICTABLE and should not occur in any real code. However, if this were to occur at the highest privilege level with interrupts disabled, it could deadlock that CPU.

Note: This erratum matches bug #5355 in the ARM internal Jira database.

Conditions

1) A load instruction is executed that accesses bytes from two different 4k pages
2) The lower page is Device or Strongly-ordered memory type
3) The upper page is Normal Write-Back Cacheable memory, that is also Read-Allocate, Write-Allocate, or both
4) The load instruction is one of the following:
    - LDRH or LDR
    - an LDM that is not 8-byte aligned
    - a VLD* to 32bit Sd registers that is not 8-byte aligned
    - a VLD* to 64bit Dd registers that is not 16-byte aligned

Implications

If the above conditions occur, it is possible that in rare cases the core will stall until the next interrupt. If this occurs in a situation where no interrupt will occur, the CPU deadlocks. Examples of where a deadlock might occur are if there is no timer interrupt in the system, or the conditions occur at the highest privilege level with interrupts disabled.

Workaround

Do not execute loads that cross between Strongly-ordered or Device memory and Normal memory pages. Architecturally, such a load is UNPREDICTABLE.

784469: CTI Authentication Status register is incorrect

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

The AUTHSTATUS register is a read-only register in the CTI that reports the debug level supported by the CTI and the current status of the debug level.

The CoreSight Architecture Specification specifies bits [3:0] in the AUTHSTATUS register as below:

- [3:2] Non-Secure Non-Invasive Debug
- [1:0] Non-Secure Invasive Debug

For each of these fields, the value of the status bits as returned by the CTI and their meanings are specified as below:

Value Description

2'b10 Functionality disabled

2'b11 Functionality enabled

In the CTI each pair of bits ([3:2] and [1:0]) in the AUTHSTATUS register currently read:

- When functionality is disabled - 2'b01

but should read (as per table above):

- When functionality is disabled - 2'b10

The bits are swapped.

Condition 1

AUTHSTATUS[1:0] - Non-secure Invasive Debug

- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Condition 2

AUTHSTATUS[3:2] - Non-secure non-Invasive Debug

- NIDEN input to the CTI is LOW
- DBGEN input to the CTI is LOW
- AUTHSTATUS register is read

Implications

The status of the debug level supported by the CTI as returned by the AUTHSTATUS register read is incorrect. The masking of trigger inputs and outputs using DBGEN and NIDEN is not affected by this erratum. The return of an incorrect value might lead to incorrect operation of debug tools.

Workaround

This is a workaround for users and tools vendors. When reading the AUTHSTATUS register, swap the bits in the affected fields and interpret the read data accordingly.

788419: If a single memory address is aliased as both shareable and non-shareable, a CMO to this same line might deadlock

Category C

Products Affected: Cortex-A15 MP Core -NEON.

Present in: r3p0, r3p1, r3p2. Fixed in r3p3.

Description

If an invalidating data cache maintenance by MVA operation (DCIMVAC or DCCIMVAC) issued by an A15 CPU hits a dirty line in the L2 cache and collides in a narrow window with a ACE snoop to the same line, the data provided as part of the snoop response might be corrupted or the system could deadlock.

The cache maintenance operation allocates a data buffer to hold the dirty data to be evicted. The snoop misses in the cache, but is detected as a hit on the eviction that is in progress. The eviction data is converted to a snoop response, and the snoop becomes associated with the dirty data in the data buffer, instead of being associated with the cache maintenance operation data buffer entry. For one cycle during this conversion when the cache maintenance operation is assigning the data buffer to the snoop, that data buffer appears to be available for an unrelated new request. If a new operation that is in the pipeline during that one cycle is allocated to the data buffer, and this allocation overwrites the data before the data is read out for the snoop response, then the snoop data is corrupted.

Note: This erratum matches bug #5378 in the ARM internal Jira database.

Conditions

This erratum requires the following sequence of conditions:

1) The same physical memory address must be aliased as both shareable and nonshareable in two different page table entries.
2) A DCIMVAC or DCCIMVAC instruction is executed on an A15 CPU to cache line A.
3) Cache line A is in the L2 cache and is dirty.
4) An ACE snoop to line A occurs.
5) The ACE snoop misses the cache, because the invalidation has already occurred, but matches on the eviction in flight before the eviction is committed to the ACE write channel.
6) The ACE snoop data response is held up, but the cache maintenance operation for cache line A is issued on the AR channel, completes, and gets an R channel response.
7) The cache maintenance operation deallocates from the L2 cache buffers before the snoop data is sent out.

Implications

The erratum conditions require the CMO to be issued after the snoop has been received, but to complete before the snoop completes. This can occur when there is disagreement on the shareability of physical address A. A15 issues a CMO to address A as non-shared memory, but another master has issued a shared memory request to A which trigger the snoop. These transactions may go through different paths in the memory system and not hazard. Disagreeing on the shareability of memory is unpredictable in the ARM architecture and therefore the erratum cannot occur with correctly programmed page tables.

Workaround

For all systems:

- Ensure that all masters in the system agree on the shareability of all memory locations and that the interconnect will not issue snoops to A15 for non-shareable memory locations.